Problems with typical array implementations of sets

Methods for set s
s.isEmpty()
s.size()
s.add(e)
s.contains(e)
s.remove(e)

A mathematical set is simply a bunch of distinct, or different, elements. The typical operations on a set s appear to the right.

A simple implementation uses an array b, with, say, the n integers occupying b[0..n-1]. We show an example with n = 5.

	0	1	2	3	4	n	
b	5	8	3	4	1		

A request to add an element involves first determining whether the element is already in b[0..n-1], because it can't be added if it's already there. Similarly, a request to remove an element involves determining whether the element is in b[0..n-1].

A search for e is typically made starting at the beginning and looking at every element until e is found —or until the end is reached, meaning e is not in the set. This takes expected-case time O(n) and worst-case time O(n), so operations add and remove take O(n) time.

	0	1	2	3	4	n	
b	1	3	4	5	8		

If the elements are from an ordered set, we could keep b[0..n-1] in ascending order and then use binary search to see whether a value is in the set. This reduces the look-up time to O(log n). However, operation add would still take expected-case and worst-case time O(n) because adding a very small value requires moving everything up one element. For example, adding 2 to (1, 3, 4, 5, 8) requires moving (3, 4, 5, 8) up one position in the array.